

Submission Worksheet

CLICK TO GRADE

<https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-milestone-1-2024/grade/jhr4>

-
-

• IT202-008-S2024 - [IT202] Milestone 1 2024

-

Submissions:

Submission Selection

1 Submission [active] 3/25/2024 7:56:37 PM



- 2.

3.

4.

- 5. Instructions

6.

7.

[^ COLLAPSE ^](#)

8.

9.

- 10. Preqs:

Go through each lesson from "Project Setup and SQL" to "User Login Enhancement" and follow the branching names while gathering the code

Merge each into Milestone1 branch

Mark the related GitHub Issues items as "done"

Implement your own custom CSS (something much different than the default "ugly" CSS given as an example)

Consider styling all forms/inputs, data output, navigation, etc

Implement JavaScript validation on Register, Logout, and Profile (include "[Client]" in the output messages to differentiate between server-side validations)

Instructions:

Make sure you're in Milestone1 with the latest changes pulled

Ensure Milestone1 has been deployed to heroku dev

Gather the requested evidence and fill in the explanations per each prompt

Save the submission and generate the output PDF

Put the output PDF into your local repository folder

add/commit/push it to GitHub

Merge Milestone1 into dev

Locally checkout dev and pull the changes

Create and merge a pull request from dev to prod to deploy Milestone1 to prod

Upload this output PDF to Canvas

Branch name: Milestone1

Tasks: 26 **Points:** 10.00

[^ COLLAPSE ^](#)

User Registration (2 pts.)

● COLLAPSE ▲

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input checked="" type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input checked="" type="checkbox"/> #3	1	Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required)
<input checked="" type="checkbox"/> #4	1	Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)
<input checked="" type="checkbox"/> #5	1	Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)
<input checked="" type="checkbox"/> #6	1	Demonstrate user-friendly message of new account being created

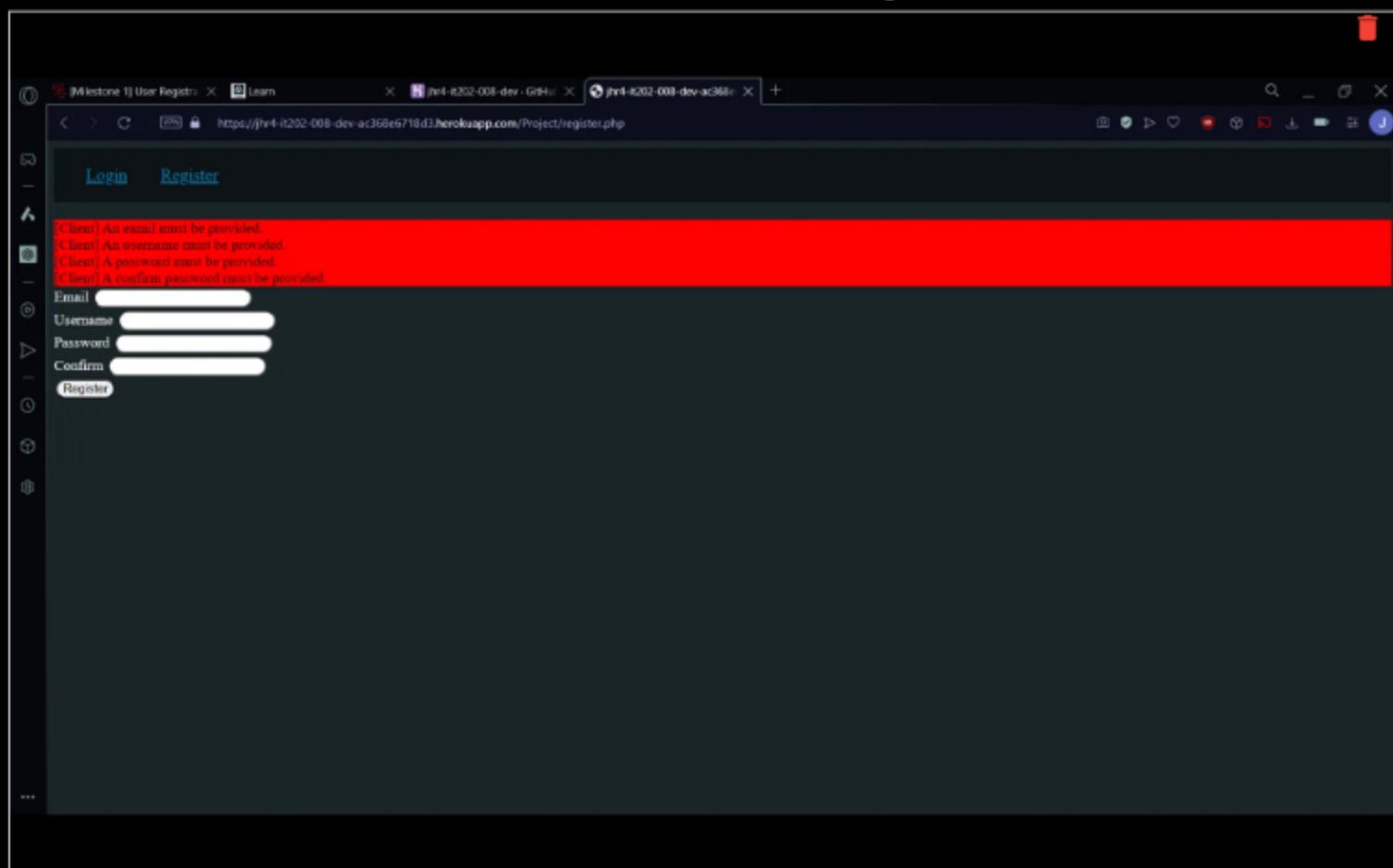
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Screenshot Demonstrates Client Side (JS) Validation for if the fields are filled in or empty.

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required)

A screenshot of a web browser window showing a registration form. The browser tabs at the top are labeled '(Milestone 1) User Registr...', 'Learn', 'jhr4-it202-008-dev - GitHub', and 'jhr4-it202-008-dev-ac368e'. The main content area shows a 'Login' and 'Register' button. Below them is a red horizontal bar containing four validation error messages: 'Client] Invalid email address.', 'Client] Username must only contain 3-16 alphanumeric characters, underscores, or dashes.', 'Client] Password too short.', and 'Client] Passwords must match.'. Below the error bar are four input fields: 'Email' (containing 'a'), 'Username' (containing 'b'), 'Password' (containing a short password), and 'Confirm' (containing a different password). A 'Register' button is located below the 'Confirm' field.

Shows JS Validation of Username, Email, and Passwords for having correct characters and size (format). Also shows the CSS.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Demonstrate JavaScript validation for each field [can be combined] (email validation (format), username validation (format), password validation (format), password and confirm password matching, and each field being required)

A screenshot of a web browser window showing a registration form. The browser tabs at the top are labeled '(Milestone 1) User Registr...', 'Learn', 'jhr4-it202-008-dev - GitHub', and 'jhr4-it202-008-dev-ac368e'. The main content area shows a 'Login' and 'Register' button. Below them is a yellow horizontal bar containing the message 'There was a problem registering.' followed by 'The chosen email is not available.' Below the bar are three input fields: 'Email' (containing 'a'), 'Username' (containing 'b'), and 'Password' (containing a password). There is no 'Confirm' field visible.

Confirm
Register

Shows email already in use message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#4 Demonstrate email already in use message (message text doesn't need to be exact, but should be clear)



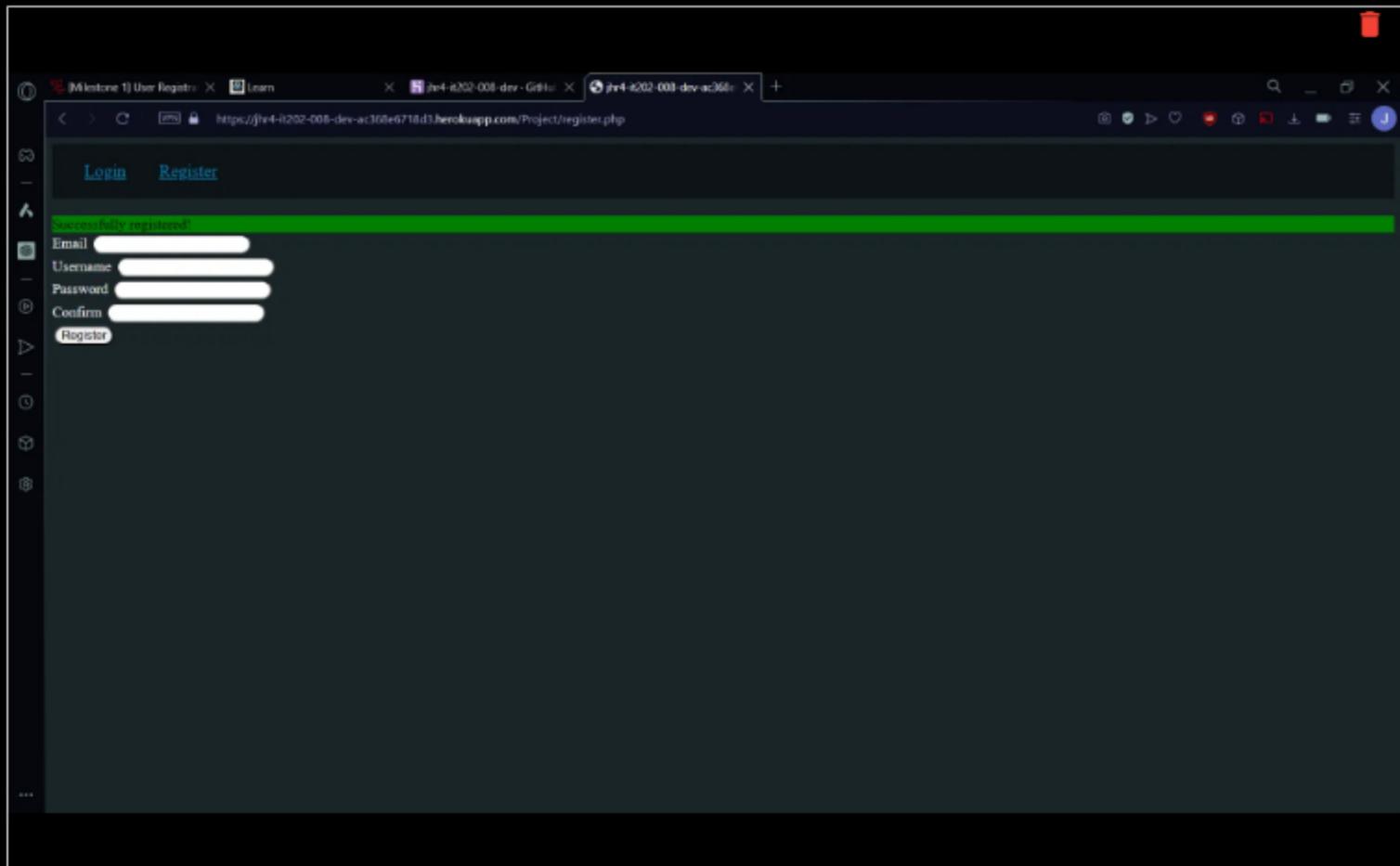
Shows username already in use message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#5 Demonstrate username already in use message (message text doesn't need to be exact, but should be clear)



Shows new account created/registered message.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#6 Demonstrate user-friendly message of new account being created

Task #2 - Points: 1

Text: Screenshot of the form code

ⓘ Details:

Should have appropriate input types for the field

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



jhr4-IT202-008



register.php X

```
1  <?php
2  require(__DIR__ . "/../../partials/nav.php");
3  reset_session();
4  ?>
5  <form onsubmit="return validate(this)" method="POST">
6      <div>
7          <label for="email">Email</label>
8          <input id="email" type="email" name="email" required/>
9      </div>
10     <div>
11         <label for="username">Username</label>
12         <input id="username" type="text" name="username" required/>
13     </div>
14     <div>
15         <label for="pw">Password</label>
16         <input type="password" id="pw" name="password" minlength="8" required/>
17     </div>
18     <div>
19         <label for="confirm">Confirm</label>
20         <input type="password" name="confirm" minlength="8" required/>
21     </div>
22     <input type="submit" value="Register" />
23 </form>
```

Form code with correct types.

Task #3 - Points: 1

Text: Screenshot of the client-side and server-side validation code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Show the JavaScript validations (include any extra files related)
<input checked="" type="checkbox"/> #2	1	Show the PHP validations (include any lib content)
<input checked="" type="checkbox"/> #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

Terminal Help ← →

jhr4-IT202-008

```
25  <script>
26      function validate(form) {
27          //email
28          if (email === ""){
29              flash("Client An email must be provided.", "danger");
30          }
31      }
32  </script>
```

```

59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```

Shows logic part of JS validation (the variables couldn't fit on ss, but they are there on top.) No extra files besides flash messages on last screenshot. Shows UCID and Date on bottom of ss.

Checklist Items (2)

#1 Show the JavaScript validations (include any extra files related)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```

// TODO 3: validate/use
$hasError = false;
if (empty($email)) {
    flash("An email must be provided.", "danger");
    $hasError = true;
}

//sanatize removes all illegal characters
$email = sanitize_email($email);

//validate
if (!is_valid_email($email)) {
    flash("Invalid email address.", "danger");
    $hasError = true;
}
if (!is_valid_username($username)) {
    flash("Username must only contain 3-16 alphanumeric characters, underscores, or dashes.", "danger");
    $hasError = true;
}
if (empty($password)) {
    flash("A password must be provided.", "danger");
    $hasError = true;
}
if (empty($confirm)) {
    flash("A confirm password must be provided.", "danger");
    $hasError = true;
}
if (!is_valid_password($password)) {
    flash("Password too short.", "danger");
    $hasError = true;
}
if (strlen($password) > 0 && $password != $confirm) {
    flash("Passwords must match.", "danger");
    $hasError = true;
}

if (!$hasError) {
    //echo "Welcome, $email";
}

// TODO 4
$hash = password_hash($password, PASSWORD_BCRYPT);           //always different output even if same password
$db = getDB();
$stmt = $db->prepare("INSERT INTO Users (email, password, username) VALUES(:email, :password, :username)");
//password & $email is placeholder; not $variables because user can put in drop table as password
try {
    $stmt->execute([':email' => $email, ':password' => $hash, ':username' => $username]);
    flash("Successfully registered!", "success");
} catch (PDOException $e) {
    #flash("There was a problem registering");
    users_check_duplicate($e->errorInfo);
}
//JHR4 - March 28, 2024
$ - #124-339 if(!$hasError)
// TODO 140 if(isset($_POST['email']) && isset($_POST['password']) && iss...

```

Checklist Items (2)**#2 Show the PHP validations (include any lib content)****#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)**

```

sanitizers.php
1  <?php
2
3  function sanitize_email($email = "") {
4      return filter_var(trim($email), FILTER_SANITIZE_EMAIL);
5  }
6
7
8  function is_valid_email($email = "") {
9      return filter_var(trim($email), FILTER_VALIDATE_EMAIL);
10 }
11
12 function is_valid_username($username) {
13     return preg_match('/^a-zA-Z\d{3,16}$/', $username);
14 }
15
16 function is_valid_password($password) {
17     return strlen($password) >= 8;
18 }
19 ?>
20
21
22
23
24 <!-- jhe4 || march 31, 2024

```



```

safer_echo.php
8  function se($v, $k = null, $default = "", $isEcho = true) {
9      if (is_array($v) && isset($k) && isset($v[$k])) {
10         $returnValue = $v[$k];
11     } else if (is_object($v) && isset($k) && isset($v->$k)) {
12         $returnValue = $v->$k;
13     } else {
14         $returnValue = $v;
15         //added 07-05-2021 to fix case where $k of $v isn't set
16         //this is to keep htmlspecialchars happy
17         if (is_array($returnValue) || is_object($returnValue)) {
18             $returnValue = $default;
19         }
20     } <- #13-20 else
21     if (!isset($returnValue)) {
22         $returnValue = $default;
23     }
24     if ($isEcho) {
25         //https://www.php.net/manual/en/function.htmlspecialchars.php
26         echo htmlspecialchars($returnValue, ENT_QUOTES);
27     } else {
28         //https://www.php.net/manual/en/function.htmlspecialchars.php
29         return htmlspecialchars($returnValue, ENT_QUOTES);
30     }
31 } <- #8-31 function se($v, $k = null, $default = "", $isEcho = true)
32 function safer_echo($v, $k = null, $default = "", $isEcho = true){
33     return se($v, $k, $default, $isEcho);
34 } // jhe4 || march 31, 2024

```

Lib content for PHP validation (safer echo to assign variable for the user inputs), and sanitizers (valid username email and password check with regex and character length. also the built in PHP sanitizer) UCID & Date on Bottom.

Checklist Items (2)**#2 Show the PHP validations (include any lib content)****#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)**

```

helpers.js
1  function flash(message = "", color = "info") {
2      let flash = document.getElementById("flash");
3      //create a div (or whatever wrapper we want)
4      let outerDiv = document.createElement("div");
5      outerDiv.className = "row justify-content-center";
6      let innerDiv = document.createElement("div");
7
8      //apply the CSS (these are bootstrap classes which we'll learn later)
9      innerDiv.className = `alert alert-${color}`;
10     //set the content
11     innerDiv.innerText = message;
12

```



```

flash_messages.php
1  <?php
2
3  function flash($msg = "", $color = "info") {
4      $message = ["text" -> $msg, "color" -> $color];
5      if (isset($_SESSION['flash'])) {
6          array_push($_SESSION['flash'], $message);
7      } else {
8          $_SESSION['flash'] = array();
9          array_push($_SESSION['flash'], $message);
10     }
11 } <- #3-11 function flash($msg = "", $color = "info")

```

```

13     outerDiv.appendChild(innerDiv);
14     //add the element to the DOM (if we don't it merely exists in memory)
15     flash.appendChild(outerDiv);
16 } <- #1-16 function flash(message = "", color = "info")
17
18 | 💡
19 // jhr4 || march 31, 2024

```

```

13     function getMessages()
14     {
15         if (isset($_SESSION['flash'])) {
16             $flashes = $_SESSION['flash'];
17             $_SESSION['flash'] = array();
18             return $flashes;
19         } <- #15-19 if (isset($_SESSION['flash']))
20         return array();
21     } <- #14-21 function getMessages()
22
23 // jhr4 || march 31, 2024

```

Flash message files for PHP and JS utilized to display the user friendly messages. UCID & Date on Bottom.

Checklist Items (3)

#1 Show the JavaScript validations (include any extra files related)

#2 Show the PHP validations (include any lib content)

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task #4 - Points: 1

Text: Screenshot of the Users table with a valid user entry

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Password should be hashed
<input checked="" type="checkbox"/> #2	1	Should have email, password, username (unique), created, modified, and id fields
<input checked="" type="checkbox"/> #3	1	Ensure left panel or database name is present (should contain your ucid)

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

The screenshot shows the MySQL Workbench interface with the following details:

- Left Panel:** Shows the database structure for 'IT202 8.0.35-Ubuntu20.04.1'. It includes a tree view of tables ('Roles', 'UserRoles', 'Users'), views, and procedures. The 'Users' table is selected.
- Properties Tab:** Shows the table structure with columns: id (int), email (varchar(100)), password (varchar(60)), created (timestamp), modified (timestamp), and username (varchar(30)).
- Query Editor:** Displays the SQL query: `SELECT * FROM "Users" LIMIT 300`.
- Results Tab:** Shows the results of the query with three rows of data:

	id	email	password	created	modified	username
1	1	hello1@test.com	\$2y\$10\$mrVR0mYUqCv.p9	2024-02-19 18:46:20	2024-03-28 17:28:48	hello1
2	2	helloworld@gmail.com	\$2y\$10\$lx0kdD6xVRyWmI	2024-03-03 19:03:03	2024-03-22 23:03:59	hello2
3	7	hello3@test.com	\$2y\$10\$SYPh1jujUYTgders	2024-03-28 16:52:04	2024-03-28 16:52:04	hello3

```
rjay@LAPTOP-PINHFC51: MINGW64 ~/OneDrive/Desktop/College/TT202/Jhr4-TT202-008 (Milestone1)
```

\$ |

Shows Users table with email, password, username (unique), created, mortified, and id fields. The passwords are hashed. UCID visible to left.

Checklist Items (3)

#1 Password should be hashed

#2 Should have email, password, username (unique), created, modified, and id fields

#3 Ensure left panel or database name is present (should contain your ucid)

Task #5 - Points: 1

Text: Explain the registration logic in a step-by-step manner from when the page loads to when the data is saved to the DB

① Details:

Don't just show code, translate things to plain English

Response:

The registration page has three stages basically: client side validation (JS & HTML), server side validation (PHP), and the mySQL part with a database.

When the page's html/css loads it displays a form with email, username, password, and confirm password. The JS and HTML validates this by checking: (1) if they are empty or not (via 'required' attribute in html and the 'return onsubmit="validate()"' which calls the JS function and returns true or false based on if the fields are === "" or not. (2) if the basic requirements/format is met via html utilizing the type="email" and "length" for password/username. Also JS validates it using regex to check if the character requirements are met or not for fields like the username and email. (3) The js code also checks if the confirm password is the same as the original.

However, as this is on the client side and can be modified by the client, there is also server side validation (php) that does the same thing as the JS, but in php there's also sanitization (removing illegal characters).

Furthermore, if the validation is met with no errors it moves onto hashing the password and calling the database to store the user information into the Users table. If there's an error during this like email has been taken, this error message is displayed to the user. Another thing to note is that in the SQL the user inputs aren't directly stored, but

rather placeholders are used to avoid SQL injections.

If any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.

Task #6 - Points: 1

Text: Include pull request links related to this feature

i Details:

Should end in /pull/#

URL #1

<https://github.com/Jhr-4/jhr4-IT202-008/pull/20>

User Login (2 pts.)

^COLLAPSE ^

Task #1 - Points: 1

Text: Screenshot of form on website page

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input type="checkbox"/> #3	1	Include correct data where username is used to login
<input type="checkbox"/> #4	1	Include correct data where email is used to login
<input type="checkbox"/> #5	1	Show success login message
<input type="checkbox"/> #6	1	Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)
<input type="checkbox"/> #7	1	Demonstrate user-friendly message of when an account doesn't exist
<input type="checkbox"/> #8	1	Demonstrate user-friendly message of when password doesn't match what's in the DB
<input type="checkbox"/> #9	1	Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)
<input type="checkbox"/> #10	1	Demonstrate session data being set (captured from server logs)

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

The screenshot shows a web browser window with a dark theme. The address bar contains the URL <https://jhr4-it202-008-dev-a366e6718d3.herokuapp.com/Project/login.php>. The page title is "Login". There are two error messages in red at the top: "Client] Email/username must be provided." and "Client] A password must be provided.". Below the errors are two input fields: "Email/Username" and "Password", both with placeholder text. A "Login" button is located below the password field.

Shows JS validation of each field being required. Shows CSS styling that's on every page too. The dev URL is on top too.

Checklist Items (3)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

This screenshot is identical to the one above, showing the same login page with validation errors and the same browser interface. It includes the red error messages, the input fields, and the "Login" button.

Shows JS validation of username and password format. Username error is shown when no @ is present.

Checklist Items (2)

#3 Include correct data where username is used to login

#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

The screenshot shows a web browser window with three tabs open. The active tab is titled 'jh4-i202-008-dev' and has the URL 'https://jh4-i202-008-dev-ac368e6718d3.herokuapp.com/Project/login.php'. The page displays a 'Login' form with two error messages at the top: 'Client Invalid Email.' and 'Client Password too short.'. Below the errors, there are input fields for 'Email/Username' and 'Password', both of which are currently empty. A 'Login' button is located below the password field. The browser interface includes a sidebar with various icons and a dark theme.

Shows JS validation of email (if it includes an "@" symbol).

Checklist Items (2)

#4 Include correct data where email is used to login

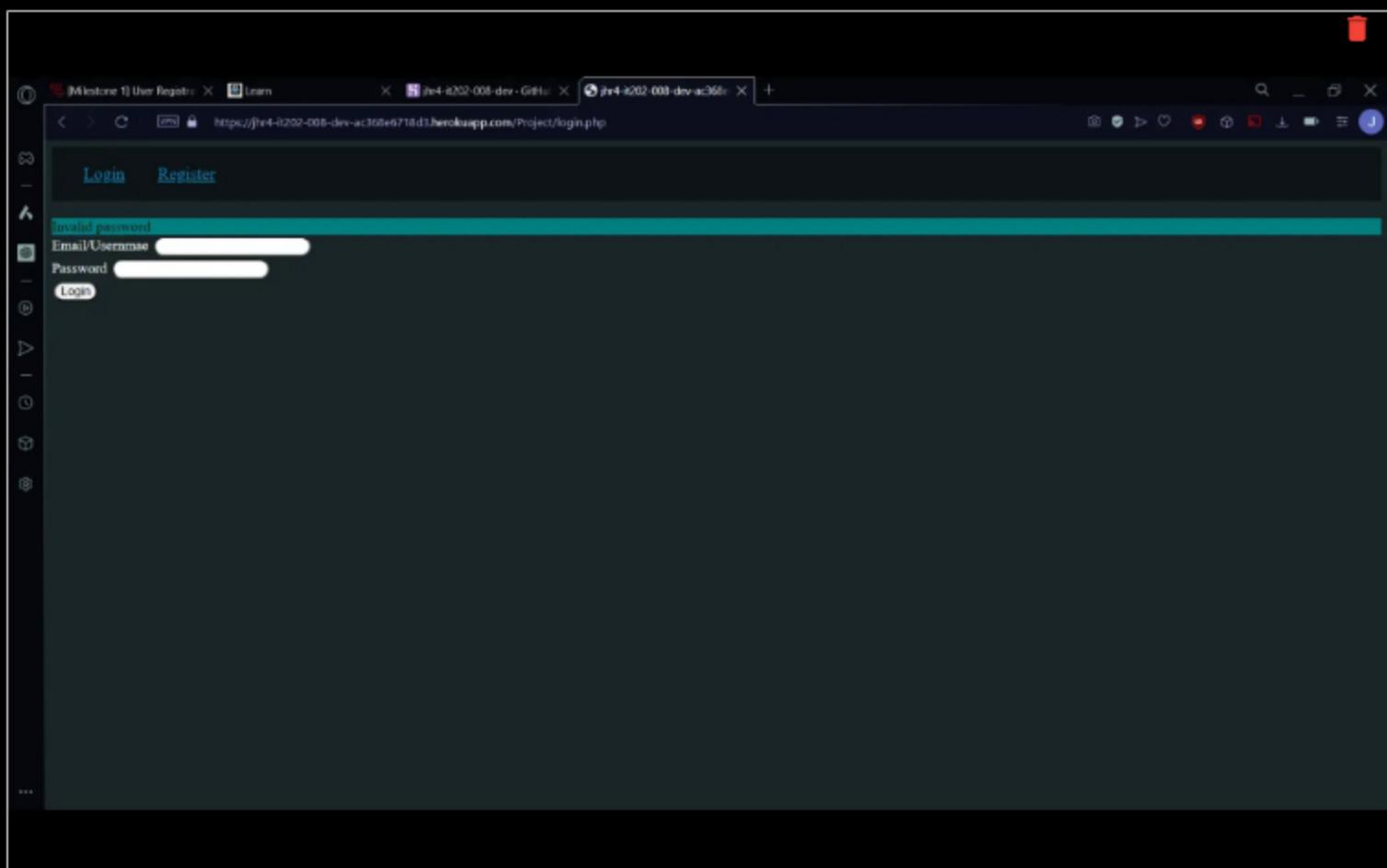
#6 Demonstrate JavaScript validation for each field [can be combined] (username format, email format, password format, and each field being required)

The screenshot shows a web browser window with three tabs open. The active tab is titled 'jh4-i202-008-dev' and has the URL 'https://jh4-i202-008-dev-ac368e6718d3.herokuapp.com/Project/login.php'. The page displays a 'Login' form with a single success message at the top: 'Email not found.' Below the message, there are input fields for 'Email/Username' and 'Password', both of which are currently empty. A 'Login' button is located below the password field. The browser interface includes a sidebar with various icons and a dark theme.

Shows message of when account doesn't exist or not found.

Checklist Items (1)

#7 Demonstrate user-friendly message of when an account doesn't exist



Shows message of password not matching what's in DB.

Checklist Items (1)

#8 Demonstrate user-friendly message of when password doesn't match what's in the DB



The screenshot shows a dark-themed web application interface. At the top, there is a navigation bar with links: Home, Profile, Create Role, List Roles, Assign Roles, and Logout. Below the navigation bar, a teal header bar displays the text "Welcome, hello". The main content area is titled "Home" and contains a large, empty space.

Shows successful login message in home page after login via a "welcome."

Checklist Items (2)

#5 Show success login message

#9 Demonstrate successful login message and the destination page (i.e., home or some landing/dashboard page)

The screenshot shows the Heroku dashboard with three tabs open: "Milestone 1] User Registr..." (active), "Learn", and "jh4-it202-008-dev-logs". The "Logs" tab is selected, showing the application logs for the "jh4-it202-008-dev-ac368" process. The logs detail a successful login attempt from IP 107.177.198.184 at 2024-09-18T17:00:00. The log entry shows the user "hello" logging in successfully, with the response body containing the message "Welcome, hello!". The logs also show other application activity and deployment details.

```
00c5-28599ae36bde Andw[47.18.6.7] dyno=web.1 connect=0ms service=0ms status=200 bytes=1400 protocol=https
2024-09-18T17:00:00.179857+00:00 heroku/router: at=info method=POST path="/Project/login.php" host=jh4-it202-008-dev-ac368e718d3.herokuapp.com request_id=(25644c2-bc46-4fc)-550d-495cc4e0 Andw[47.18.6.7] dyno=web.1 connect=0ms service=0ms status=202 bytes=2475 protocol=https
2024-09-18T17:00:00.179538+00:00 app[web.1]: 107.1.0.184 - - [18/Nov/2024:17:00:00 +0000] "POST /Project/login.php HTTP/1.1" 202 2373 "https://jh4-it202-008-dev-ac368e718d3.herokuapp.com/Project/login.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/121.0.0.0" 0:0:0.0
2024-09-18T17:00:00.179145+00:00 app[web.1]: [2024-11-18T17:00:00.17:00:00 UTC] Session ended array (
2024-09-18T17:00:00.179145+00:00 app[web.1]: "flash" =>
2024-09-18T17:00:00.179171+00:00 app[web.1]: array (
2024-09-18T17:00:00.179171+00:00 app[web.1]:   0 =>
2024-09-18T17:00:00.179171+00:00 app[web.1]:     array (
2024-09-18T17:00:00.179171+00:00 app[web.1]:       "text" => "Welcome, hello",
2024-09-18T17:00:00.179171+00:00 app[web.1]:       "color" => "info",
2024-09-18T17:00:00.179171+00:00 app[web.1]:     ),
2024-09-18T17:00:00.179171+00:00 app[web.1]:   ),
2024-09-18T17:00:00.179171+00:00 app[web.1]: "user" =>
2024-09-18T17:00:00.179171+00:00 app[web.1]:   array (
2024-09-18T17:00:00.179171+00:00 app[web.1]:     "id" => 1,
2024-09-18T17:00:00.179171+00:00 app[web.1]:     "email" => "helloguest.com",
2024-09-18T17:00:00.179171+00:00 app[web.1]:     "username" => "hello",
2024-09-18T17:00:00.179171+00:00 app[web.1]:     "role" =>
2024-09-18T17:00:00.179171+00:00 app[web.1]:       array (
2024-09-18T17:00:00.179171+00:00 app[web.1]:         0 =>
2024-09-18T17:00:00.179171+00:00 app[web.1]:           array (
2024-09-18T17:00:00.179171+00:00 app[web.1]:             "name" => "Admin",
2024-09-18T17:00:00.179171+00:00 app[web.1]:           ),
2024-09-18T17:00:00.179171+00:00 app[web.1]:         ),
2024-09-18T17:00:00.179171+00:00 app[web.1]:       ),
2024-09-18T17:00:00.179171+00:00 app[web.1]:     ),
2024-09-18T17:00:00.179171+00:00 app[web.1]:   ),
2024-09-18T17:00:00.179171+00:00 app[web.1]: )
2024-09-18T17:00:00.179171+00:00 app[web.1]: 107.1.0.184 - - [18/Nov/2024:17:00:00 +0000] "GET /Project/home.php HTTP/1.1" 200 2348 "https://jh4-it202-008-dev-ac368e718d3.herokuapp.com/Project/home.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/121.0.0.0" 0:0:0.0
ac368e718d3.herokuapp.com/Project/login.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/121.0.0.0" 0:0:0.0
2024-09-18T17:00:00.179171+00:00 app[web.1]: [2024-11-18T17:00:00.17:00:00 UTC] Session ended array (
2024-09-18T17:00:00.179171+00:00 app[web.1]: "flash" =>
```

Autoscroll with output

heroku.com Blogs Careers Documentation Support Terms of Service Privacy Cookies © 2024 Salesforce.com

Checklist Items (1)

#10 Demonstrate session data being set (captured from server logs)

Task #2 - Points: 1

Text: Screenshot of the form code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)
<input checked="" type="checkbox"/> #2	1	Show JavaScript validations (include any extra files related)
<input checked="" type="checkbox"/> #3	1	Show PHP validations (include any lib content)
<input checked="" type="checkbox"/> #4	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task Screenshots:

Gallery Style: Large View

[Small](#) [Medium](#) [Large](#)

```

4   <form onsubmit="return validate(this)" method="POST">
5     <div>
6       <label for="email">Email/Username</label>
7       <input type="text" name="email" required/>
8     </div>
9     <div>
10    <label for="pw">Password</label>
11    <input type="password" id="pw" name="password" minlength="8" required/>
12  </div>
13  <input type="submit" value="Login"/>
14 </form>
15 |   <!-- jhr4 || march 28, 2024-->
16 <script>
```

Shows HTML form code with two fields: Email/Username and Password. The type for the first field is text because it can be username too. ucid/date on bottom.

Checklist Items (2)

#1 Should have proper input types for the fields (note in the caption if you're using two fields for Username/Email or a combined field)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```
<!-- jhr4 || march 28, 2024-->
<script>
    function validate(form) {
        //TODO 1: implement JavaScript validation
        //ensure it returns false for an error and true for success
        let password = form.password.value;
        let email_user = form.email.value;

        let isValid = true;

        //email or username
        if (email_user === ""){ //if empty it must be provided
            flash("[Client] Email/username must be provided.", "danger");
            isValid = false;
        } else if (/@/.test(email_user)){ //not empty from earlier & if has @ -> check email regex conditions
            if ( !/^([a-zA-Z0-9_.%]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})$/.test(email_user)){
                flash("[Client] Invalid Email", "danger");
                isValid = false;
            }
        } else if (!/^([a-zA-Z0-9_.-]{3,16})$/.test(email_user)){ //not empty from earlier & no @email -> check username regex conditions
            flash("[Client] Invalid Username", "danger"); //user should already know format...
            isValid = false;
        }

        //password
        if (password === ""){
            flash("[Client] A password must be provided.", "danger");
            isValid = false;
        }
        if (password !== "" && password.length < 8){
            flash("[Client] Password too short.", "danger");//will potentially be changed to just "Invalid Password." User should know correct length.
            isValid = false;
        }

        return isValid;
    } <- #17-50 function validate(form)
</script>
```

JS validation code. ucid/date on top.

Checklist Items (2)

#2 Show JavaScript validations (include any extra files related)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```
//TODO 3
$hasError = false;
if (empty($email)) {
    flash("Email must not be empty");
    $hasError = true;
}
if (str_contains($email, "@")) {
//sanitize
//filter_var($email, FILTER_SANITIZE_EMAIL);
$email = sanitize_email($email);

//validate
/*if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    flash("Invalid email address");
    $hasError = true;
}*/
if (!is_valid_email($email)) {
    flash("Invalid email address");
    $hasError = true;
}
} else {
    if (!is_valid_username($email)) {
        flash("Invalid username");
    }
}
```

```

        $hasError = true;
    }
} <- #77-82 else

if (empty($password)) {
    flash("password must not be empty");
    $hasError = true;
}
if (strlen($password) < 8) {
    flash("Password too short");
    $hasError = true;
} //jhr4 || march 28, 2024
if (!hasError) {
    //Final
}

```

PHP validation (format & existence) UCID and date on bottom.

Checklist Items (0)

```

login.php ✘
$db = getDB();
$stmt = $db->prepare("SELECT id, email, username, password from Users where email = :email or username = :email");
try {
    $r = $stmt->execute([":email" => $email]);
    if ($r) {
        $user = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($user) {
            $hash = $user["password"];
            unset($user["password"]);
            if (password_verify($password, $hash)) {
                $_SESSION["user"] = $user;
                try {
                    //lookup potential roles
                    $stmt = $db->prepare("SELECT Roles.name FROM Roles
JOIN UserRoles on Roles.id = UserRoles.role_id
where UserRoles.user_id = :user_id and Roles.is_active = 1 and UserRoles.is_active = 1");
                    $stmt->execute([":user_id" => $user["id"]]);
                    $roles = $stmt->fetchAll(PDO::FETCH_ASSOC); //fetch all since we'll want multiple
                } catch (Exception $e) {
                    error_log(var_export($e, true));
                }
                //save roles or empty array
                if (isset($roles)) {
                    $_SESSION["user"]["roles"] = $roles; //at least 1 role
                } else {
                    $_SESSION["user"]["roles"] = []; //no roles
                }
                flash("Welcome, " . get_username());
                die(header("Location: home.php"));
            } else {
                flash("Invalid password");
            }
        } else {
            flash("Email not found");
        }
    } <- #100-129 if ($user)
} catch (Exception $e) {
    flash("<pre>" . var_export($e, true) . "</pre>"); //jhr4 || march 28, 2024
}

```

Validation with database (making sure email/username exists in db and password matches to one stored in DB) ucid & date on bottom

Checklist Items (2)

#3 Show PHP validations (include any lib content)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

<code>sanitizers.php</code>	<code>safer_echo.php</code>
<pre> 1 <?php 2 3 function sanitize_email(\$email = "") { 4 return filter_var(trim(\$email), FILTER_SANITIZE_EMAIL); 5 } </pre>	<pre> 8 function se(\$v, \$k = null, \$default = "", \$isEcho = true) { 9 if (is_array(\$v) && isset(\$k) && isset(\$v[\$k])) { 10 \$returnValue = \$v[\$k]; 11 } else if (is_object(\$v) && isset(\$k) && isset(\$v->\$k)) { 12 \$returnValue = \$v->\$k; 13 } else { 14 \$returnValue = \$default; 15 } 16 17 if (\$isEcho) { 18 echo \$returnValue; 19 } else { 20 return \$returnValue; 21 } 22 } </pre>

Same as register. user_helpers isn't shown as that's not apart of validation it occurs after login and storing data into session. Lib content for PHP validation (safer echo to assign variable for the user inputs), and sanitizers (valid username email and password check with regex and character length. also the built in PHP sanitizer) UCID & Date on Bottom.

Checklist Items (2)

#3 Show PHP validations (include any lib content)

#4 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Same as register. Flash message files for PHP and JS utilized to display the user friendly messages. UCID & Date on Bottom.

Checklist Items (0)



[^COLLAPSE^](#)

Task #3 - Points: 1

Text: Explain the login logic step-by-step from when the page loads to when the data is fetched from the DB and stored in the session

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Don't just show code, translate things to plain English
<input checked="" type="checkbox"/> #2	1	Explain how the session works and why/how it's used

Response:

The login logic is similar to the register page for the validation, however this time the email/username field is validated based on the condition of if it has an @ or not which tells us if the user is trying to put an email or username. After this the validation is basically the same and was explained earlier.

After again similar to before, we access the database's Users table. This time however, we select the email or username (based on what the user put) by using a placeholder of the input. We also select the password corresponding to that email/user. If there's no data for that email we return email not found. After this we check the password using PHP's "password_verify" method using the hashed password from the db (which is unset) and the users typed in password.

If the user was found and the passwords match, the users data array from the db is stored in the session and also the roles information is gathered of the user and stored in the session. The session was actually loaded when the person loaded up the page as the navbar is required. The navbar utilizes the session_start() method in php. This looks for the PHPSESSID cookie on the client side and sees if that session exists on the server side if there is it will load it and if not it will create a new session. The cookie is also created with the navbar being loaded and has a domain & path of only the project website meaning it only exists on the website and path and not others. Basically it means the user data is transferred across pages in this website. This is important as the session is a way to verify that the user has logged in allowing the user to use other pages that should be locked for someone that's not logged in like home or profile. The idea of roles being in the session is also important as it allows for other pages to be unlocked like admin ones if they have the role for it.

In the end if everything is successful the login page's script is killed (with die) and user is redirected to home page. If any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.



[^COLLAPSE^](#)

Task #4 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

<https://github.com/Jhr-4/jhr4-IT202-008/pull/21/>

User Logout (1 pt.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Capture the following screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Screenshot of the navigation when logged in (site)
<input checked="" type="checkbox"/> #2	1	Screenshot of the redirect to login with the user-friendly logged-out message (site)
<input checked="" type="checkbox"/> #3	1	Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

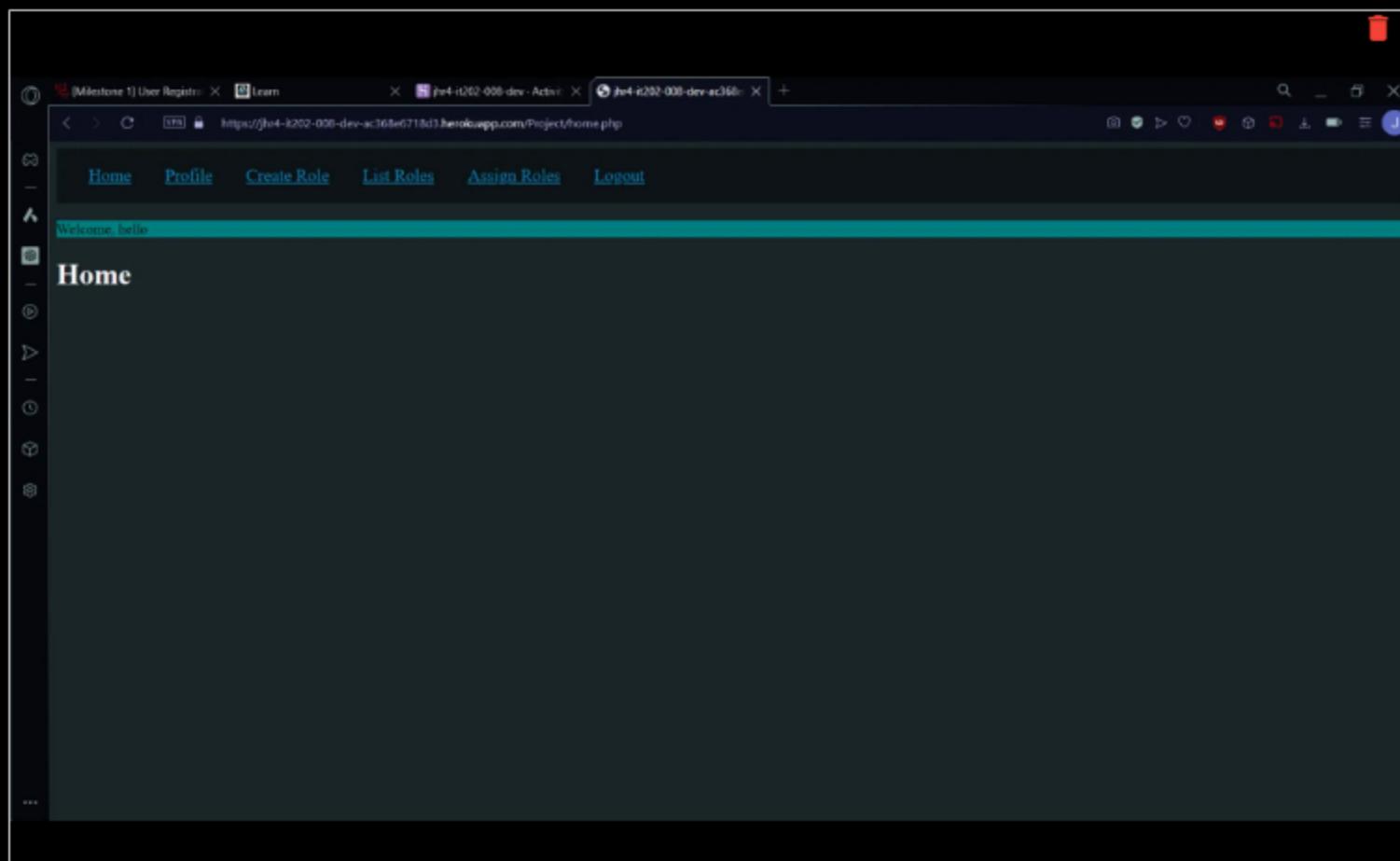
Task Screenshots:

Gallery Style: Large View

Small

Medium

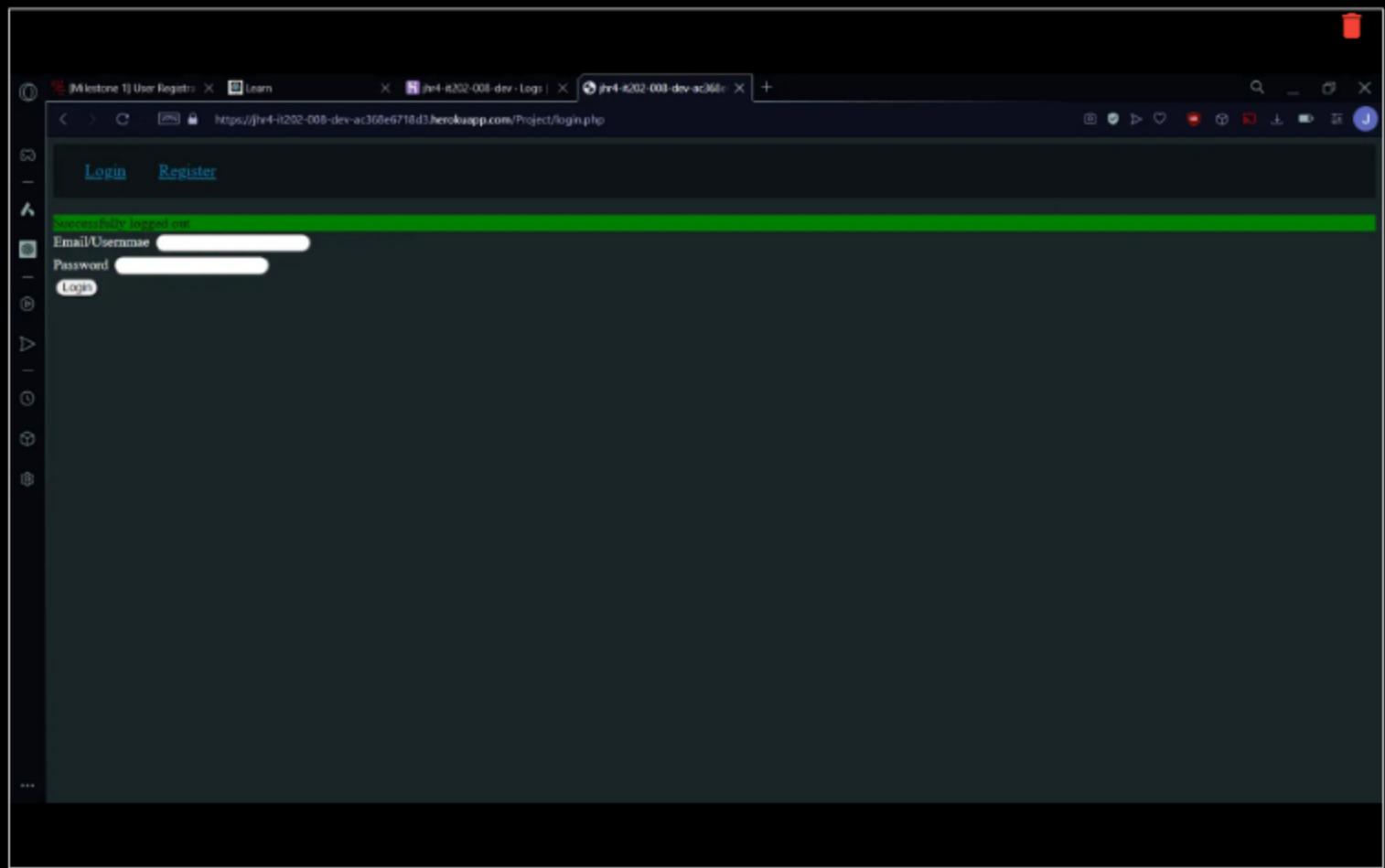
Large



Shows nav bar of admin account login.

Checklist Items (1)

#1 Screenshot of the navigation when logged in (site)



Shows Logout message and being redirected to Login page.

Checklist Items (1)

#2 Screenshot of the redirect to login with the user-friendly logged-out message (site)

A screenshot of a code editor with two tabs open. The left tab is named "logout.php" and contains the following PHP code:

```
1 <?php
2 session_start();
3 require(__DIR__ . "/../../lib/functions.php");
4 reset_session();
5
6 flash("Successfully logged out", "success");
7 header("Location: login.php");
8 ?>
9
10 <!-- jhr4 || march 28, 2024|
```

The right tab is named "reset_session.php" and contains the following PHP code:

```
1 <?php
2 function reset_session()
3 {
4     session_unset();
5     session_destroy();
6     session_start();
7 } <- #3-7 function reset_session()
```

Shows how session is destroyed with the function `reset_session()` which is also shown on the right. ucid / date on bottom.

Checklist Items (1)

#3 Screenshot of the logout-related code showing the session is destroyed (code). Ensure ucid/date comment is present.

Task #2 - Points: 1

Text: Include pull request links related to this feature

● Details:

Should end in /pull/#

URL #1

<https://github.com/Jhr-4/jhr4-IT202-008/pull/21>

● Basic Security Rules and Roles (2 pts.)

[^COLLAPSE ^](#)

Task #1 - Points: 1

Text: Authentication Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Screenshot of the function that checks if a user is logged in
<input checked="" type="checkbox"/> #2	1	Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on
<input checked="" type="checkbox"/> #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
<input checked="" type="checkbox"/> #4	1	Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



[Login](#) [Register](#)

You must be logged in to view this page

Email/Username

Password

Shows message of logged out user trying to access login protected page.

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a login-protected page while being logged out

```
7 function is_logged_in($redirect = false, $destination = "login.php")
8 {
9     $isLoggedIn = isset($_SESSION["user"]);
10    if ($redirect && !$isLoggedIn) {
11        //if this triggers, the calling script won't receive a reply since die()/exit() terminates it
12        flash("You must be logged in to view this page", "warning");
13        die(header("Location: $destination"));
14    } <- #10-14 if ($redirect && !$isLoggedIn)
15    return $isLoggedIn;
16 } //jhr4 || march 28, 2024 <- #8-16 function is_logged_in($redirect = false, $destination = "logi...
```

Shows function to check if user is logged in. UCID & date on bottom.

Checklist Items (2)

#1 Screenshot of the function that checks if a user is logged in

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

A screenshot of a code editor showing two files: register.php and profile.php. The profile.php file is open and contains the following code:

```
1 <?php
2 require_once(__DIR__ . "/../../partials/nav.php");
3 is_logged_in(true); //jhr4 || march 28, 2024
4 ?>
```

The code editor has a search bar at the top right containing "jhr4-IT202-008".

Shows login check function being used in profile page. The function directly used on: profile.php and home.php. It's also used in the has_role() function so its also used on: assign_roles.php, create_role.php, and list_roles.php. UCID & Date on Bottom

Checklist Items (2)

#2 Screenshot of the login check function being used (i.e., profile likely). Also caption what pages it's used on

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task #2 - Points: 1

Text: Authorization Screenshots

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Screenshot of the function that checks for a specific role
<input checked="" type="checkbox"/> #2	1	Screenshot of the role check function being used. Also caption what pages it's used on
<input checked="" type="checkbox"/> #3	1	Include ucid/date code comments in all screenshots (one per screenshot is sufficient)
		Demonstrate the user-friendly message of trying to manually access a role-protected

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Shows message of user trying to access a role protected page while being logged out.

Checklist Items (1)

#4 Demonstrate the user-friendly message of trying to manually access a role-protected page while being logged out

```
17 function has_role($role) {  
18     if (is_logged_in() && isset($_SESSION["user"]["roles"])) {  
19         foreach ($_SESSION["user"]["roles"] as $r) {  
20             if ($r["name"] === $role) {  
21                 return true;  
22             }  
23         } //19-23 foreach ($_SESSION["user"]["roles"] as $r)  
24     } //18-24 if (is_logged_in() && isset($_SESSION["user"]["r  
25     return false;
```

A screenshot of a code editor displaying a PHP function named 'has_role'. The function checks if the user is logged in and if their session array contains a 'user' key with a 'roles' value. It then iterates through the 'roles' array to see if any role matches the '\$role' parameter. If a match is found, it returns true; otherwise, it returns false. The code is color-coded for readability, with different colors used for keywords, functions, and variable names.

```
26 } //jhr4 || march 28, 2024 <- #17-26 function has_role($role)
```

Shows function to check for a specific role in user_helpers.php file. ucid date on bottom.

Checklist Items (2)

#1 Screenshot of the function that checks for a specific role

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

```
assign_roles.php X user_helpers.php
1 <?php
2 //note we need to go up 1 more directory
3 require(__DIR__ . "/../../partials/nav.php");
4
5 if (!has_role("Admin")) {
6     flash("You don't have permission to view this page", "warning");
7     die(header("Location: $BASE_PATH" . "/home.php"));
8 } //jhr4 march 28, 2024
```

Shows role check function being used. It's used on: assign_roles.php, create_role.php, and list_roles.php UCID/date on bottom

Checklist Items (2)

#2 Screenshot of the role check function being used. Also caption what pages it's used on

#3 Include ucid/date code comments in all screenshots (one per screenshot is sufficient)

Task #3 - Points: 1

Text: Screenshots of UserRoles and Roles Tables

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Points	Details
■ #1	1	At least one valid and enabled User->Role reference (UserRoles table)
■ #2	1	UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns
■ #3	1	Roles Table should have id, name, description, is_active, modified, and created columns
■ #4	1	At least one valid and enabled Role (Roles table)
■ #5	1	Ensure left panel or database name is present in each table screenshot (should contain your ucid)

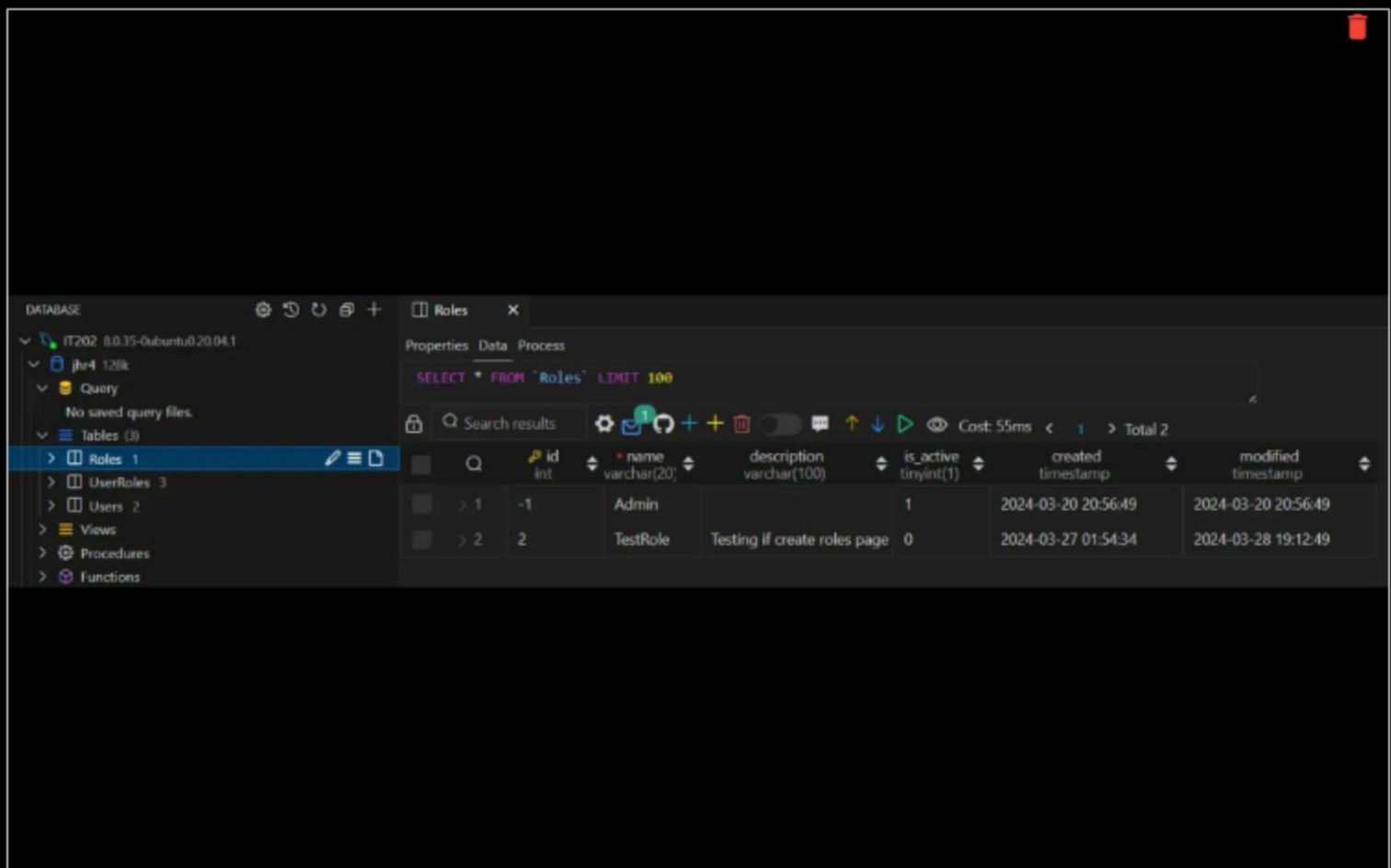
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Shows roles table with admin role enabled and TestRole disabled. All 6 of these columns are present: id, name, description, is_active, modified, and created. UCID on top left.

Checklist Items (3)

#3 Roles Table should have id, name, description, is_active, modified, and created columns

#4 At least one valid and enabled Role (Roles table)

#5 Ensure left panel or database name is present in each table screenshot (should contain your ucid)

	Q	id int	user_id int	role_id int	is_active tinyint(1)	created timestamp	modified timestamp
	> 1	1	1	-1	1	2024-03-20 20:58:45	2024-03-20 20:58:45
	> 2	2	1	2	0	2024-03-27 01:54:58	2024-03-27 01:56:49
	> 3	3	2	2	0	2024-03-27 01:54:58	2024-03-27 01:56:49

User Roles table with user_id 1 with active admin role (-1). Table contains all these columns: id, user_id, role_id, is_active, created, and modified columns. UCID on top left.

Checklist Items (3)

#1 At least one valid and enabled User->Role reference (UserRoles table)

#2 UserRoles Table should have id, user_id, role_id, is_active, created, and modified columns

#5 Ensure left panel or database name is present in each table screenshot (should contain your ucid)

Task #4 - Points: 1

Text: Explain how Roles and UserRoles tables work in conjunction with the Users table

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	What's the purpose of the UserRoles table?
<input checked="" type="checkbox"/> #2	1	How does Roles.is_active differ from UserRoles.is_active?

Response:

The Roles table is just a list of roles in which each role name and id is unique. The UserRoles table is a combination of the Users table and Roles table in which users are given roles by utilizing their user_id and basically matching it with the role_id from the Roles table. Basically Roles table establishes the roles while UserRoles assigns the roles.

Roles.is_active for the actual role while UserRoles.is_active is for the user. If the Roles.is_active = 1 it's active and it

Roles.is_active for the actual role while UserRoles.is_active is for the user. If the Roles.is_active is 1 it's active and it works, but if it's 0 it's basically disabled. For UserRoles.is_active it's per user so if a role is toggled for a user they don't have the role. This system basically allows roles to be "deleted" from users or the whole website. "deleted" basically just means disabled as the actual role is never removed just toggled active or inactive.

Task #5 - Points: 1

Text: Include pull request links related to this feature

i Details:

Should end in /pull/#

URL #1

<https://github.com/Jhr-4/jhr4-IT202-008/pull/27>

User Profile (2 pts.)

[COLLAPSE](#)

Task #1 - Points: 1

Text: View Profile Website Page

Checklist

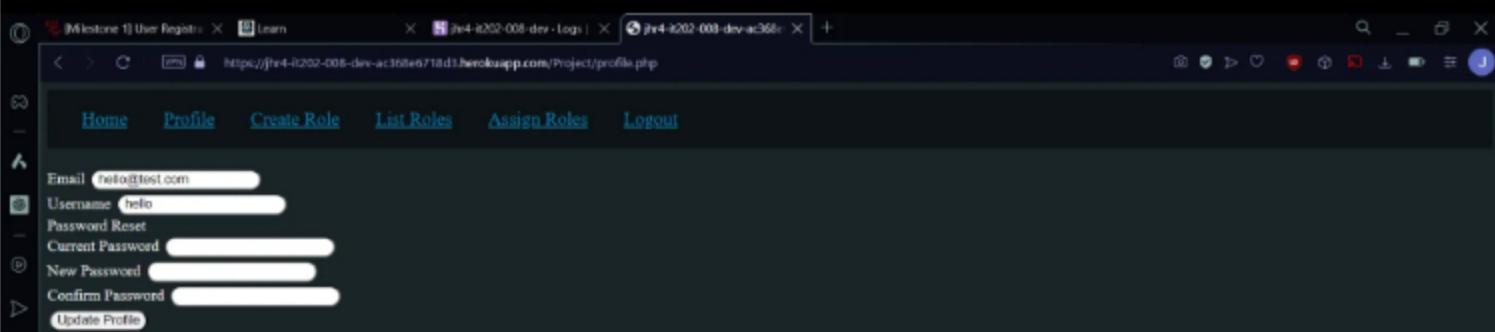
*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input checked="" type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input checked="" type="checkbox"/> #3	1	Show the profile form correctly populated on page load (username, email)
<input checked="" type="checkbox"/> #4	1	Should have the following fields: username, email, current password, new password, confirm password (or similar)

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Shows profile page with decent CSS of inputs being rounded. There are 5 fields: email, username, current password, new password, confirm password. The username and email fields are prefilled on load. CSS will be made better when adding in Bootstraps next milestone.

Checklist Items (4)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

#3 Show the profile form correctly populated on page load (username, email)

#4 Should have the following fields: username, email, current password, new password, confirm password (or similar)

Task #2 - Points: 1

Text: Explain the logic step-by-step of how the data is loaded and populated when the profile page is visited

Details:

Don't just show code, translate things to plain English

Response:

When the profile page is visited the data that is loaded/populated is the email and username. This is done through php which utilizes the `get_user_email()` and `get_username()` function which placing it ad the value for the fields. These functions work by getting the username and email from the Session which is stored if the person is logged in.

The session is set when the user visits the website and logs in which then places the username email and roles of the user in the session which is loaded in every page thru the nav bar.

Task #3 - Points: 1

[COLLAPSE](#)

TASK #3 | POINTS: 1

Text: Edit Profile Website Page

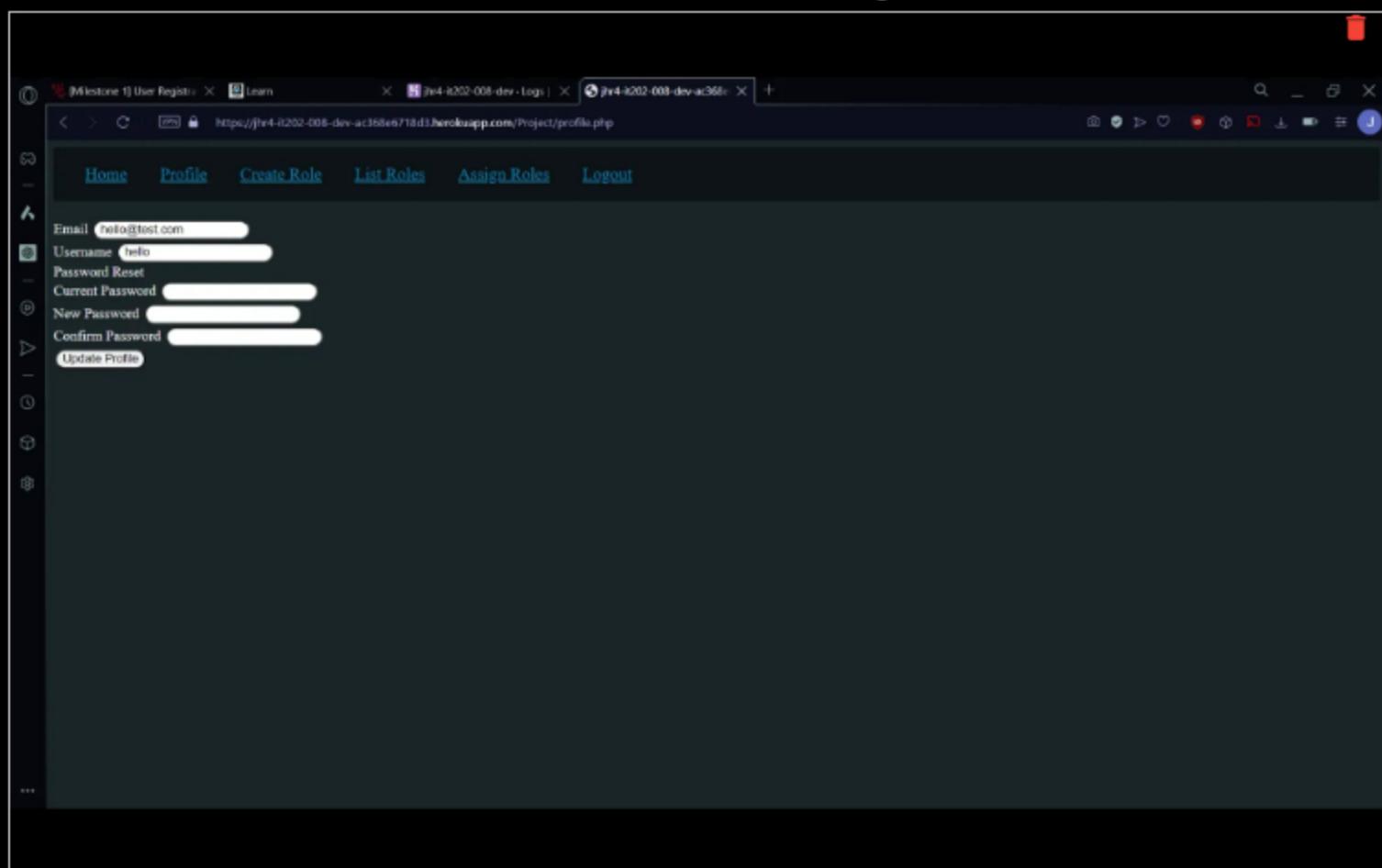
Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Heroku dev url should be present in the address bar
<input checked="" type="checkbox"/> #2	1	Should have thoughtful CSS applied
<input checked="" type="checkbox"/> #3	1	Demonstrate with before and after of a username change (including success message)
<input checked="" type="checkbox"/> #4	1	Demonstrate with a before and after of an email change (including success message)
<input checked="" type="checkbox"/> #5	1	Demonstrate the success message of updating password
<input checked="" type="checkbox"/> #6	1	Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)
<input checked="" type="checkbox"/> #7	1	Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

Task Screenshots:

Gallery Style: Large View

[Small](#)[Medium](#)[Large](#)

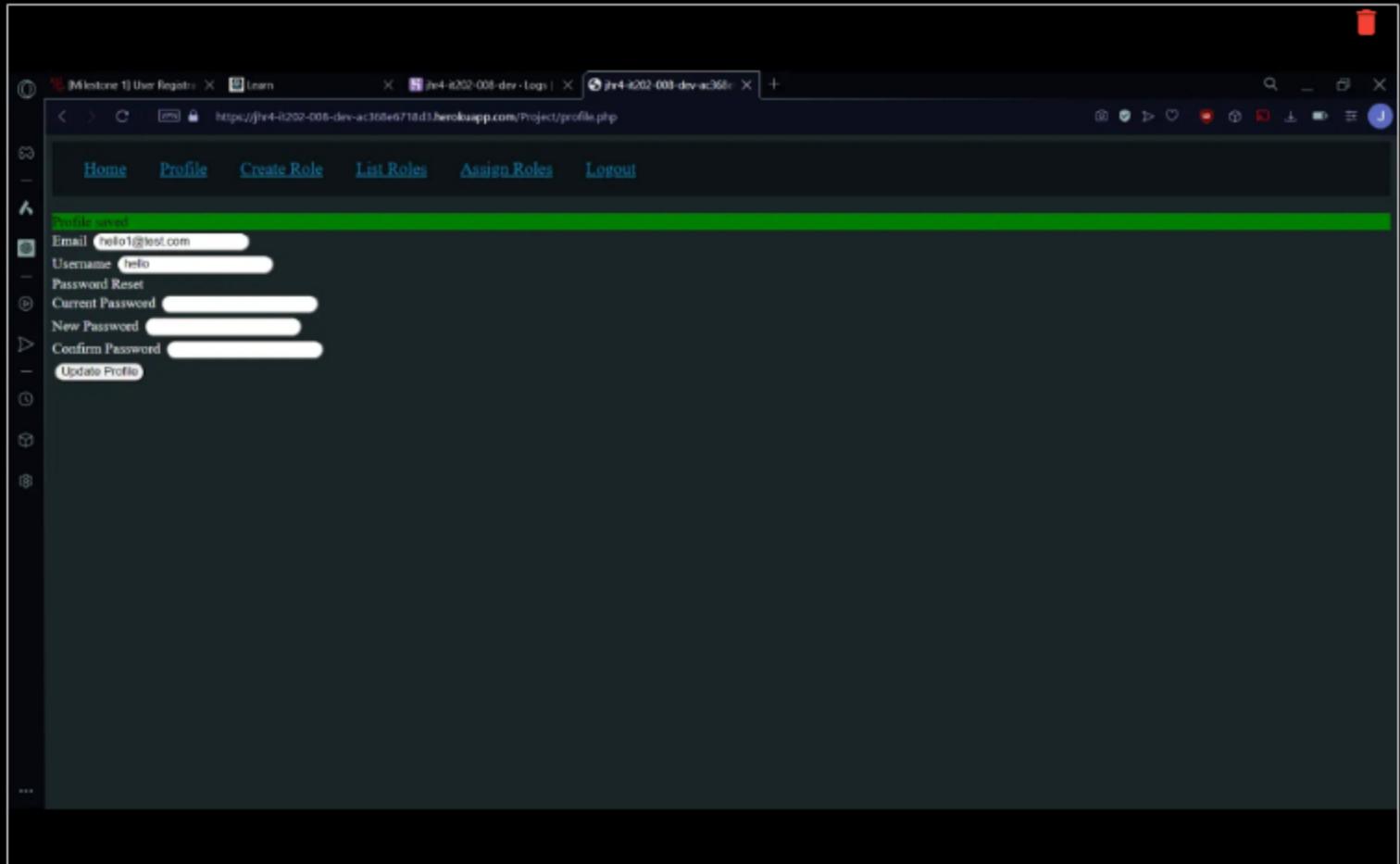
Before image of profile page with loaded username & email. With CSS styling and Heroku dev link on top.

Checklist Items (4)

#1 Heroku dev url should be present in the address bar

#2 Should have thoughtful CSS applied

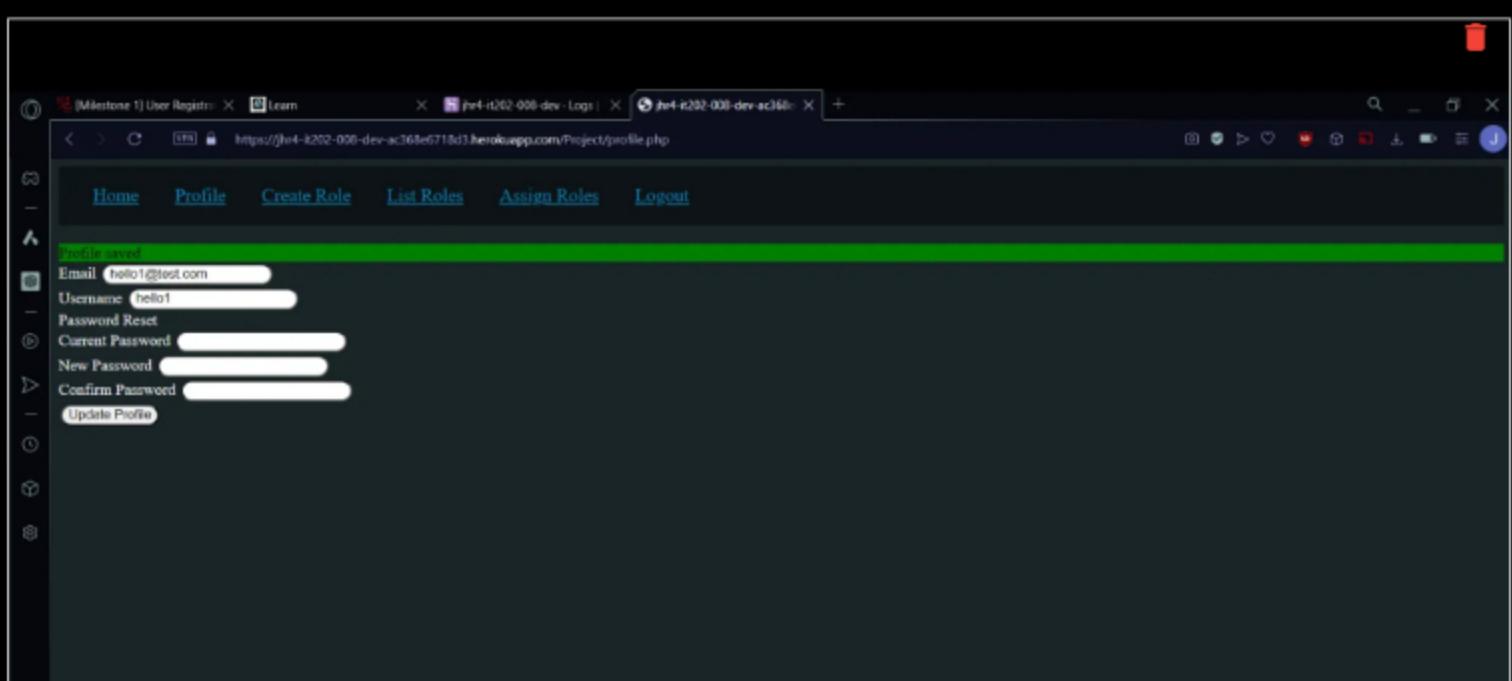
#3 Demonstrate with before and after of a username change (including success message)



After screenshot of email being changed, success message, "profile saved."

Checklist Items (1)

#4 Demonstrate with a before and after of an email change (including success message)



After screenshot of username being changed success image, "profile saved."

Checklist Items (1)

#3 Demonstrate with before and after of a username change (including success message)

Profile saved
Password reset

Email hello1@test.com
Username hello1
Password Reset
Current Password
New Password
Confirm Password

Update Profile

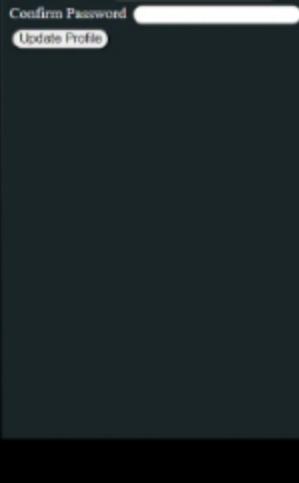
Shows success message of password being updated, "password reset."

Checklist Items (1)

#5 Demonstrate the success message of updating password

Client] An email must be provided.
Client] An username must be provided.
Client] Current password is invalid.
Client] A new password must be provided.
Client] A confirm password must be provided.

Email
Username
Password Reset
Current Password
New Password



Screenshot showing JS validation if all fields are empty. (The current password is filled in because at least one of the password fields must be filled in to show validation of __ password must be filled. If this isn't done it would show not filled in even if the user didn't try to password reset.)

Checklist Items (1)

#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)

The screenshot shows a user profile edit page with several validation errors displayed in a red bar:

- [Client] Invalid email address.
- [Client] Username must only contain 3-16 alphanumeric characters, underscores, or dashes.
- [Client] Current password is invalid.
- [Client] Password and Confirm password must match.
- [Client] New password too short.

The form fields include:

- Email: [redacted]
- Username: [redacted]
- Password Reset: [redacted]
- Current Password: [redacted]
- New Password: [redacted]
- Confirm Password: [redacted]

Buttons: Update Profile

Shows JS format validation for all fields. Invalid password just means too short (user should know what's wrong).

Checklist Items (1)

#6 Demonstrate JavaScript user-friendly validation messages (email format, username format, password format, new password matching confirm password)



The chosen email is not available.
Current password is invalid

Email:

Username:

Password Reset

Current Password:

New Password:

Confirm Password:

Shows email is not available/already in use. Shows current password is invalid/doesn't match what's in DB.

Checklist Items (1)

#7 Demonstrate PHP user-friendly validation messages (desired email is already in use, desired username is already in use, current password doesn't match what's in the DB)

Task #4 - Points: 1

Text: Explain the logic step-by-step of how the data is checked and saved for the following scenarios

Checklist

*The checkboxes are for your own tracking

#	Points	Details
<input checked="" type="checkbox"/> #1	1	Updating Username/Email
<input checked="" type="checkbox"/> #2	1	Updating password
<input checked="" type="checkbox"/> #3	1	Don't just show code, translate things to plain English

Response:

Username/Email:

Data is checked for the username/email with HTML, JS, and PHP validation. The basic idea is the same as the registration page where the HTML and JS are client side. HTML validates the email by utilizing type="email" which makes sure the user types in a valid email. Then to enhance the validation JS is used with regex to check for things like if the email has valid characters, a "@" character, a domain after it followed by a . and a top level domain. The username also uses regex to make sure it has only lowercase alphanumeric characters, _ and - and meets the length of 3-16. While this data is prefilled, the JS and PHP still verify that the user didn't try to change it to empty "" and alerts them that it must be filled in. The PHP validation is also the same to the JS and basically in a different language.

After this format validation comes calling the database and checking for duplicate emails and usernames which is done by a try catch in which PHP/mysql is used to create a placeholder of the input and then executing an update to the database table. If it doesn't work it displays the error like a duplicate email or user. The update of the database

the database table. If it doesn't work it displays the error like a duplicate email or user. The update of the database occurs thru the user id which is already stored in the session.

Password:

For the password there's also HTML, JS and PHP validation. The HTML validation makes sure all the passwords are length of 8. The JS makes sure that if the user changes/inputs into any field of password (current, new, or confirm), it will validate it. Firstly it checks that all the fields are filled in and aren't just empty, "" if so there is a user friendly message displayed. Furthermore, the length of the new passwords are also checked and it's made sure that the length is at least 8 and the confirm password is matched with the new password to make sure no typo occurred by user. After this the PHP if no errors the users id is used to fetch their current password which is compared to what the user inputs for current password (as a placeholder). Then, if this goes correctly the new password is hashed and updated in the database Users table.

For both if any validation fails or there's an error an user friendly message is displayed (using the created flash function) for the user to try again.

Task #5 - Points: 1

Text: Include pull request links related to this feature

Details:

Should end in /pull/#

URL #1

<https://github.com/Jhr-4/jhr4-IT202-008/pull/25>

Misc (1 pt.)

Task #1 - Points: 1

Text: Screenshot of wakatime

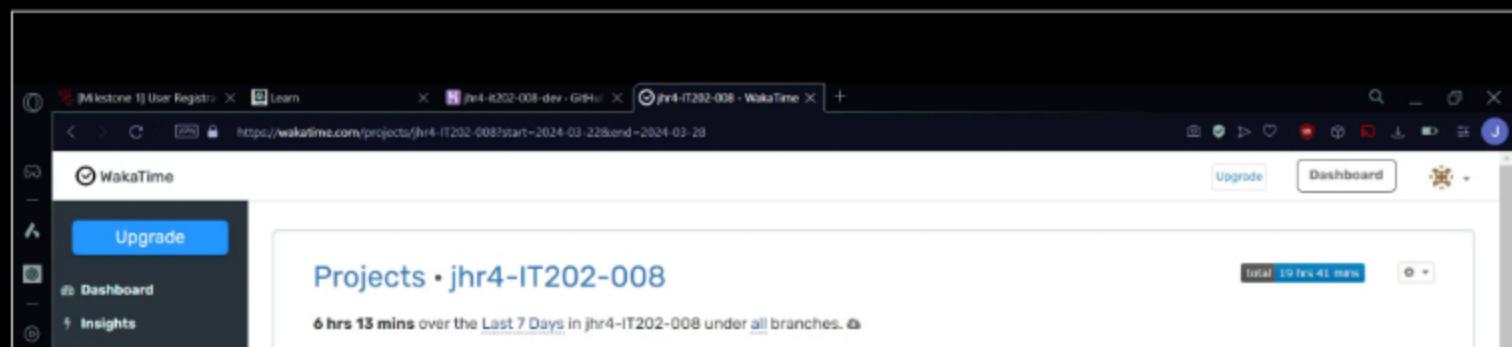
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large





Shows Wakatime of this project.

Task #2 - Points: 1

Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

The GitHub Project Board shows the following tasks across three columns:

- Todo:** This item hasn't been started.
- In Progress:** This item is actively being worked on.
- Done:** This item has been completed.

Completed tasks (in the Done column) include:

- MST1 - User will be able to register a new account
- MST1 - User will be able to login to their account (given they enter the correct credentials)
- MST1 - User will be able to logout
- MST1 - Basic security rules implemented
- MST1 - Basic Roles implemented
- MST1 - Site should have basic style/theme applied; everything should be styled
- MST1 - Any output messages/errors should be "User friendly"
- MST1 - User will be able to see their profile
- MST1 - User will be able to edit their profile

Task #3 - Points: 1

Text: Provide a direct link to the project board on GitHub

URL #1

<https://github.com/users/Jhr-4/projects/1>

Task #4 - Points: 1

Text: Provide a direct link to the login page from your prod instance

URL #1

<https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/login.php>

End of Assignment